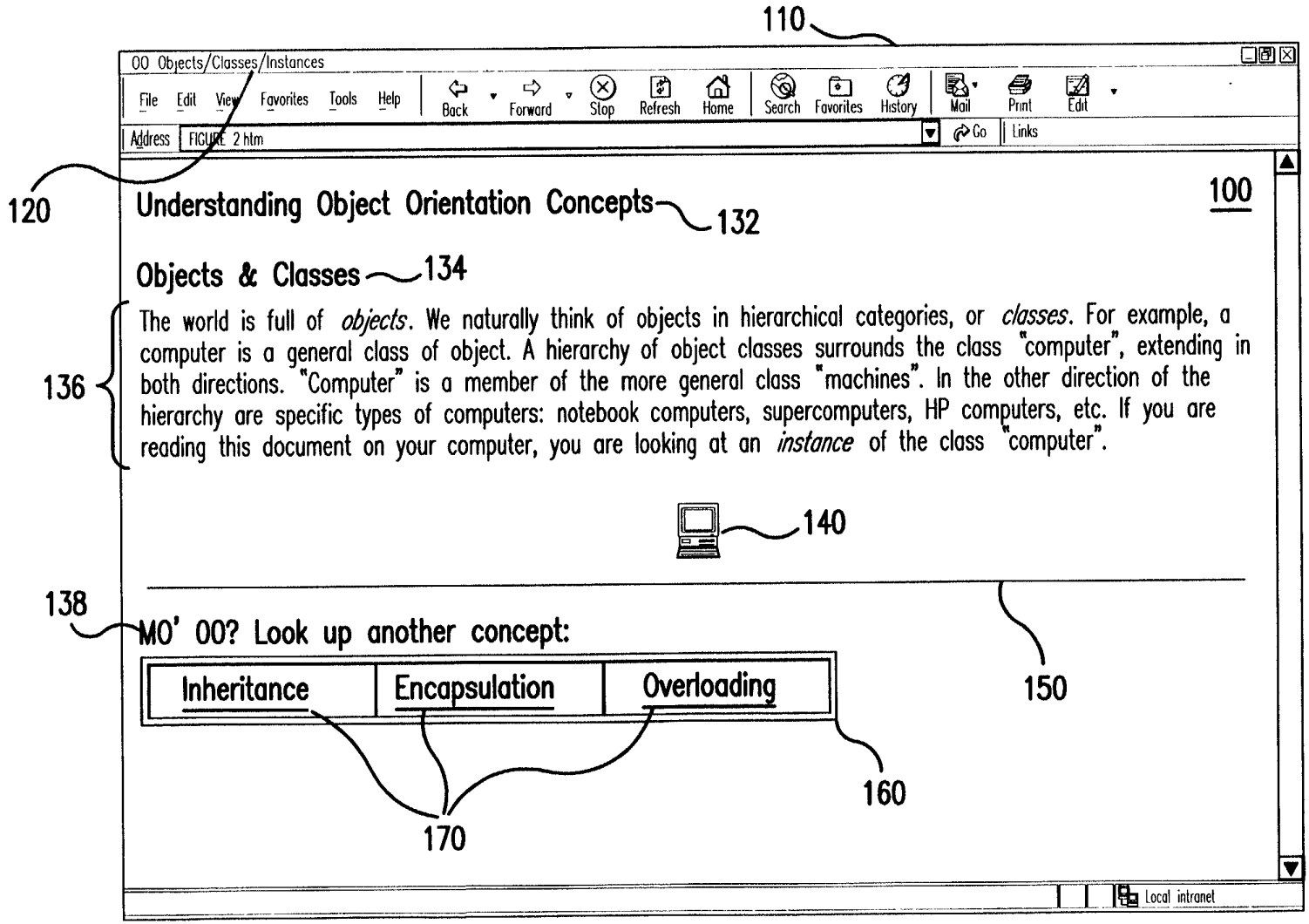

FIG.1

200

```
<HTML>
<HEAD>
<TITLE>00 Objects/Classes/Instances</TITLE>                    220
</HEAD>
<BODY>

<P><FONT color="forestgreen" size+"+2" id="arial">
   <B>Understanding Object Orientation Concepts</B>            232
</FONT></P>

<P><FONT color="black">
<B>Objects &amp; Classes</B>                                   234
</FONT></P>

<P>The world is full of <I>objects</I>. We naturally think of
objects in hierarchical categories, or <I>classes</I>. For
example, a computer is a general class of object. A hierarchy
of object classes surrounds the class &quot;computer&quot;,
extending in both directions. &quot;Computer&quot; is a
member of the more general class &quot;machines&quot;. In       236
the other direction of the hierarchy are specific types of
computers: notebook computers, supercomputers, HP
computers, etc. If you are reading this document on your
computer, you are looking at an <I>instance</I> of the class
&quot;computer&quot;.</P>

<P><CENTER>
  <IMG src="computer.gif" border="0">                          240
</CENTER></P>
<HR />                                                         250

Mo'00?  Look up another concept:<BR>                           238

<TABLE border+"2" width+60%">
<TR>
  <TD>
  <A href="http://www.mooo.org/inh.htm">Inheritance</A>
  <TD>
  <TD>
  <A href="http://www.mooo.org/encap.htm">Encapsulation</A>    270
  </TD>
  <TD>
  <A href="http://www.mooo.org/overld.htm">Overloading</A>
  </TD>
</TR>
</TABLE>

</BODY>
</HTML>
```

260

Fig. 2

300

```
main ()
{
    htmlDocument* document = new htmlDocument (stdout,
                                    "00 Objects/Classes/Instances");
310 tableGrid*      table     = new tableGrid (1, 0, 0, "60%");
    centered*       center    = new centered ();

    document->add( new paragraph () );
332 document->add( new htmlText ("Understanding Object Orientation
                Concepts", "forestgreen", normal, bold, "+2", "arial"));

    document->add( new paragraph () );
334 document->add( new htmlText ("Objects & Classes", "black",
                                                    normal, bold));

    explanation = query(oo_concept_database, concept);
    find_first_and_italicise(explanation_text, "object", "class",
336                                                    "instance");
    document->add( new paragraph() );
    document->add( new htmlText(explanation_text) );

    document->add( new paragraph() );
340 center->add( new image(explanation_image) );
    document->add(center);

350 document->add( new horizontalRule() );

338 document->add( new htmlText("Mo' OO?  Look up another link:") );

    table->newRow();
    table->addField( new anchor("http://www.mooo.com/Inheritance",
                        new htmlText ("Inheritance") ) );
360 table->addField( new anchor("http://www.mooo.com/Encapsulation",
                        new htmlText ("Encapsulation") ) );
    table->addField( new anchor("http://www.mooo.com/Overloading",
                        new htmlText ("Overloading") ) );
    document->add(table);

    delete document;
}
```

Fig. 3

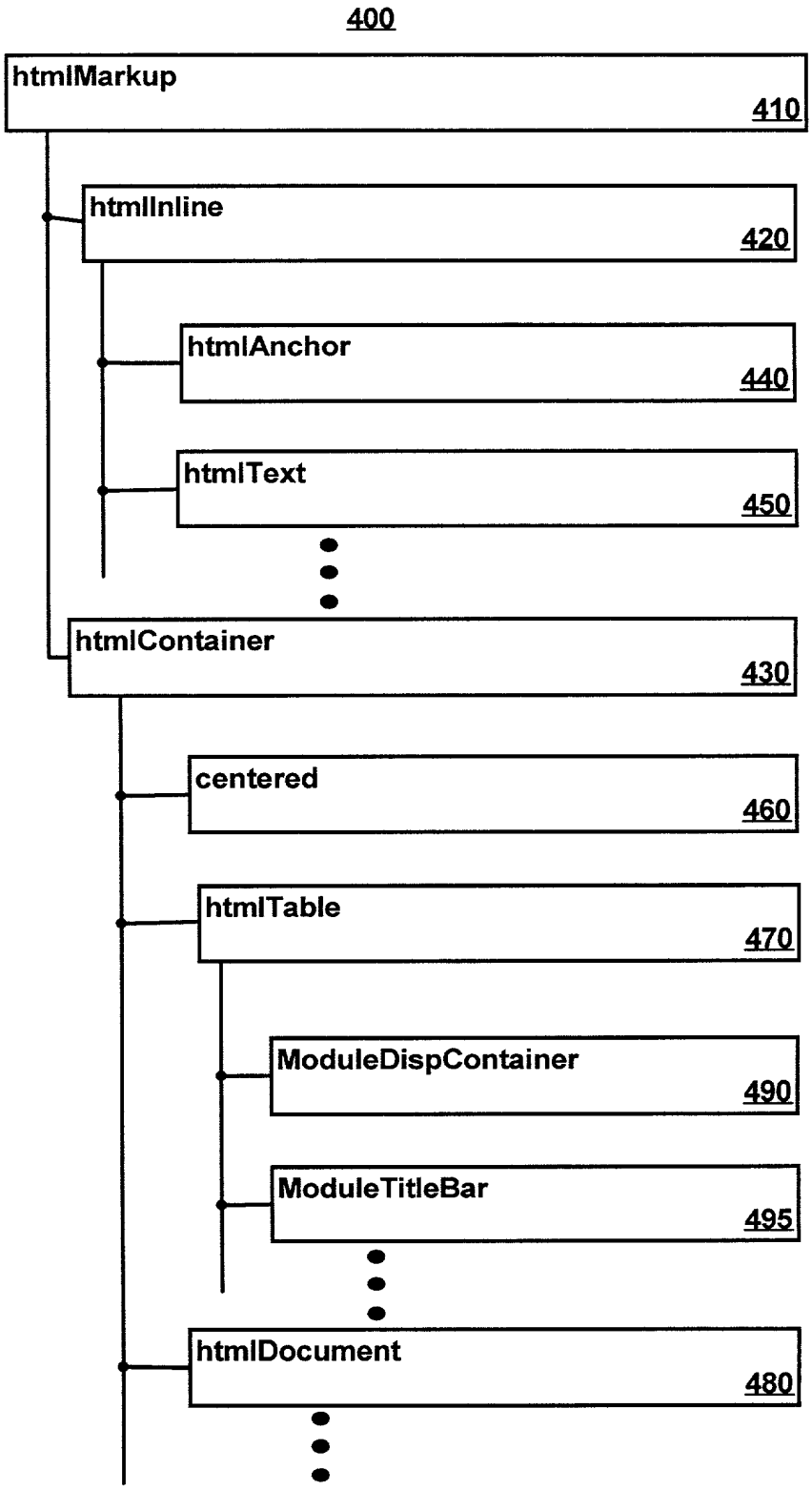
400
htmlMarkup
410
htmlInline
420
htmlAnchor
440
htmlText
450
htmlContainer
430
centered
460
htmlTable
470
ModuleDispContainer
490
ModuleTitleBar
495
htmlDocument
480

Fig. 4

410

```
// This class is an interface for defining the basic HTML/XML
// relationship between a child element and its parent.

class htmlMarkup
{
protected:
        htmlMarkup* parent = NULL;
        FILE*       fptr   = NULL;
public:
        htmlMarkup();
        virtual ~htmlMarkup();
        virtual setParent(htmlMarkup* parent)                    510
                { this.parent = parent }
}
```

Fig. 5

420

```
class htmlInline extends htmlMarkup
{
protected:
        DynamicArray* buffer = NULL;
public:
        htmlInline();
        virtual ~htmlInline()
                { if (buffer) fprintf(parent.fptr, "%s", buffer)}    610
}
```

Fig. 6

440

```
class htmlAnchor extends htmlInline
{
public:
        htmlAnchor (String href, htmlMarkup* label)              710
                { buffer   = "<a";
                  buffer += " href="+href;
                  buffer += ">";

                  // flush the label markup to this buffer
                  label.setParent(this);
                  delete label;
                  buffer += "</a>";
                }
}
```

Fig. 7

430

```
class htmlContainer extends htmlMarkup
{
protected:
        FILE*fptr = NULL;
public:
        htmlContainer();
        virtual ~htmlContainer()
                { if (fptr && parent.fptr)
                        concatencateFiles(fptr, parent.fptr);}
}
```

Fig. 8

450

```
class htmlTable extends htmlContainer
{
public:
        htmlTable()
                { fptr = new temporaryFile();
                  print("<table>");
                }

        vitrual ~htmlTable ()
                { print("</table>"); }

        void addRow()
                { print("<tr>");     }

        void addContent(htmlMarkup* content)
                { print("<td>");

                  // flush the child content to this table
                   content.setParent(this);
                  delete content;

                   print("</td>");
                }
}
```

Fig. 9

1000

Class	Style	HTML element
commentText	htmlInline	<!-- -- >
htmlText	htmlInline	ASCII text
formattedText	htmlInline	<PRE>
embeddedText	htmlInline	<LAYER>
htmlImage	htmlInline	<IMG>
htmlAnchor	htmlInline	<A>
paragraph	htmlInline	<P>
centered	htmlContainer	<CENTER>
lineBreak	htmlInline	
noLineBreak	htmlInline	<NOBR>
horizontalRule	htmlInline	<HR>
table	htmlContainer	<TABLE>
htmlDocument	htmlContainer	<HTML>
htmlForm	htmlContainer	<FORM>
formInput	htmlInline	<INPUT>
formTextReadOnly	htmlInline	<TEXT>
selectionList	htmlContainer	<SELECTION>

Fig. 10

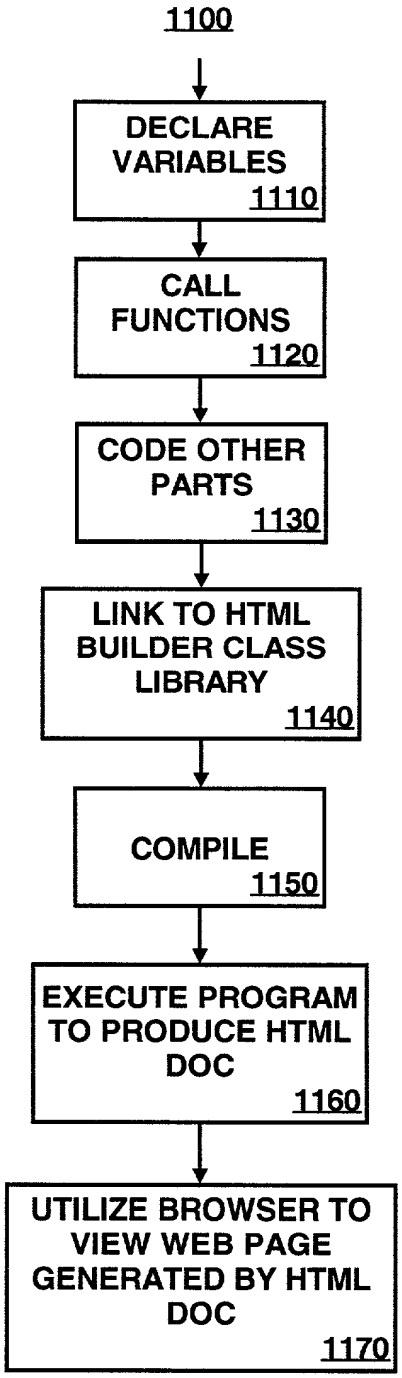
1100
DECLARE VARIABLES
1110
CALL FUNCTIONS
1120
CODE OTHER PARTS
1130
LINK TO HTML BUILDER CLASS LIBRARY
1140
COMPILE
1150
EXECUTE PROGRAM TO PRODUCE HTML DOC
1160
UTILIZE BROWSER TO VIEW WEB PAGE GENERATED BY HTML DOC
1170

Fig. 11